Sveučilište J.J. Strossmayera u Osijeku
Filozofski fakultet
Preddiplomski studij informacijskih znanosti

Zrna Kojčić

Pregled algoritama sortiranja

Završni rad

Mentor : doc.dr.sc. Gordana Dukić
Komentor: mr.sc. Anita Papić

Osijek, 2012.

Sadržaj

Sažetak:

U ovom radu bit će opisana problematika sortiranja algoritama. Na početku će biti naveden povijesni kontekst nastanka algoritama općenito kao i njihova svrha, razvoj i primjena u razvoju programskih sustava. Nakon upoznavanja s pojmom algoritama, bit će pojašnjena primjena djelatnosti sortiranja podataka i opisana svrha iste. Nakon upoznavanja s pojmovima vezanih uz sortiranje algoritama bit će objašnjena razlika algoritama prema složenosti odnosno brzini koja je važna pri sortiranju velikog broja podataka. Nadalje u poglavljima će se dati pregled nekih od jednostavnih algoritama kao što su sortiranje izborom, sortiranje umetanjem te sortiranje zamjenom susjednih elemenata ili mjehuričasto sortiranje te opisan način na koji funkcionira svaki od navedenih algoritama. Nakon pregleda jednostavnih algoritama sortiranja pojasnit će se pojam rekurzije te će biti dan pregled algoritama za sortiranje koji su zasnovani na rekurziji, primjerice to su algoritmi sortiranje spajanjem, (engl. *merge sort*) i brzo sortiranje, (engl. *quick sort*). Također, bit će objašnjen pojam rekurzije kao metode definiranja funkcija u kojima se definirajuća funkcija primjenjuje unutar definicije. Na kraju rada će biti objašnjeno sortiranje pomoću binarnog stabla te će biti navedeni primjeri algoritama zasnovanih na binarnom stablu kao što su sortiranje obilaženjem binarnog stabla traženja, engl. *tree sort* i sortiranje pomoću hrpe, engl. *heap sort*. U svakom poglavlju će biti opisani načini te dani primjeri kako svaki od navedenih algoritama funkcionira te implementacija u programskom jeziku C i analiza vremenske složenosti nekih od navedenih algoritama sortiranja.

Ključne riječi: algoritmi, sortiranje, vremenska složenost, rekurzija.

1. Uvod

Računalnim algoritmima smatramo niz naredbi koji efikasno rješavaju neki problem. Iako se pojam algoritama u zadnje vrijeme najčešće povezuje s računalima, bitno je znati da je algoritam nastao mnogo ranije, prvenstveno u matematici. Prvi poznati algoritam nastao je još u 3. stoljeću prije Krista, a poznat je kao Euklidov algoritam. Euklid je taj algoritam opisao u svojoj knjizi „Elementi". Riječ „algoritam" dolazi od latinskog prijevoda imena iranskog matematičara Al-Hvarizmija koji se smatra ocem algebre jer je definirao osnovna pravila rješavanja linearnih i kvadratnih jednadžbi.

Poznavanje algoritama jest temelj uspješnog programiranja, a najviše u razvoju složenih programskih sustava. Izrada algoritma u programiranju prethodi samom pisanju programa. Pri pisanju programa najvažnije je znati što se zapravo od njega očekuje. Poput rješavanja zadataka u svim drugim područjima potrebno je znati kako postaviti problem. Kada je problem ispravno postavljen potrebno je izraditi skicu rješenja, a zatim se izrađuju algoritmi.

Algoritmi za sortiranje pojavljuju se već u najranijim danima razvoja računala.

Računalo je samo po sebi odličan alat za pohranu i obradu podataka. „*BubbleSort*" algoritam je nastao 1956. godine, a „*Quicksort*", jedan od najbržih i najčešće korištenih algoritama danas, razvijen je još 1960. godine. Danas se primjena sortiranja podataka može vidjeti u određenim radnjama na računalu, kao i u poslovima s velikim bazama podataka u raznim firmama. Iz osnovnih algoritama za sortiranje, s napretkom tehnologije razvili su se brži i efikasniji algoritmi za sortiranje, a u ovome radu bit će dan kratak pregled vrsta algoritama sortiranja te usporedba složenosti.

Sortiranjem podataka podrazumijevamo djelatnost uređenja određenog niza redanjem elemenata prema određenim kriterijima, a glavno pitanje koje si trebamo postaviti pri sortiranju jest koji su to kriteriji. U početku sortiranja u računalnom svijetu koristili su se jednostavni algoritmi kao što je sortiranje izborom. Ovaj algoritam u listi pronalazi najmanji element i on mijenja mjesto s prvim elementom u listi. U jednostavne algoritme spadaju također mjehuričasto sortiranje te sortiranje umetanjem. Pri mjehuričastom sortiranju prolazi se redom po elementima liste i svaki se od njih uspoređuje sa svojim sljedbenikom. Sortiranje umetanjem radi na principu

dijeljenja liste na dva dijela: prvi koji je sortiran i drugi koji će se sortirati. Nadalje u radu dolazimo do rekurzivnih algoritama. To su algoritmi koji pozivaju same sebe sve dok ne postignu određeni uvjet. U rekurzivne algoritme spadaju sortiranje spajanjem i brzo sortiranje. Sortiranje spajanjem (engl. *merge sort*) je prvi sa složenosti manjom od kvadratne, a algoritmi s ovom vrstom složenosti najčešće koriste metodu *podijeli pa vladaj*. Kod brzog sortiranja također se koristi metoda *podijeli pa vladaj*, no kod njega je faza spajanja potpuno izbjegnuta. Nakon rekurzivnih algoritama, dolazimo do algoritama sortiranja koji se temelje na binarnom stablu. U radu će biti opisana dva algoritma za sortiranje ove vrste: sortiranje pomoću hrpe i sortiranje obilaskom binarnog stabla. Sortiranje pomoću hrpe (engl. *heap sort*) temelji se na posebnom tipu podataka – hrpi, a hrpom nazivamo binarno stablo sa svojstvom da vrh ima manju vrijednost od svoja dva nastavka. Sortiranje obilaskom binarnog stabla traženja vrlo je slično sortiranju pomoću hrpe, ono u čemu se razlikuju je vrsta binarnog stabla koje se koristi. U slučaju sortiranja obilaskom binarnog stable traženja je to binarno stablo traženja, a u slučaju sortiranja pomoću hrpe je to hrpa. U nastavku rada svaki od navedenih algoritama sortiranja će biti dodatno objašnjeni.

2. Sortiranje podataka

Sortiranje elemenata je preuređenje nekog niza tako što su elementi poredani prema određenom kriteriju, bilo u rastućem ili u opadajućem redoslijedu.

Potreba za sortiranjem podataka se često javlja, najbolji primjeri za to su rječnici, telefonski imenici, knjige u knjižnicama, popisi studenata na fakultetima i sl.

Najvažnije pitanje koje si pri sortiranju moramo postaviti jest po kojem kriteriju se sortiraju podaci. Ako blok podataka, tj. slog ima ključ, onda se isti koristi za definiranje poretka u listi na kojoj je zadano ***n*** slogova za sortirati.

Postoje razni algoritmi koji se međusobno razlikuju po složenosti odnosno brzini. Brzina je važna kada moramo sortirati veliki broj podataka. Dolazimo do pitanja usporedbe brzine sortiranja. To se radi na dva načina; mjerenjem vremena i usporedbom broja operacija koje program obavlja. Taj broj operacija je jedna od mjera složenosti algoritma.

Kod brojeva je prilično intuitivno jasno što znači sortiranje, dok kod tekstualnih podataka (stringova) se podaci obično sortiraju po tzv. leksikografskom odnosno abecednom poretku:

AAA, AAB, AAC...

Problem sortiranja u znanosti je još uvijek aktualan te se traže gotovo bilo kakve mogućnosti ubrzanja, tako da se nedavno došlo do otkrića algoritma „intro-sort“ koji je kombinacija dvaju već poznatih algoritma za sortiranje, a sam je brži od svakog pojedinačno.[1] Najniža klasa algoritama za sortiranje obuhvaća najjednostavnije, ali i najsporije algoritme kao što su to primjerice algoritmi *„bubble-sort”* ili *„selection-sort”* i *„insertion sort“*, dok u najvišu klasu spadaju algoritmi poput *„quick-sort”*,*“heap sort“ i „merge-sort”*.

[1] Baumgartner, Alfonzo; Poljak, Stjepan. Sortiranje podataka. //Osječki matematički list 5 (2005), str. 21-28.

3. Sortiranje izborom

Kao što je već u radu navedeno u jednostavne algoritme sortiranja spada sortiranje izborom (engl. *selection sort*). Ovaj algoritam radi sljedeće: u listi se pronalazi najmanji element i on mijenja mjesto s prvim elementom u listi. Nakon toga, prvi će sadržavati onu vrijednost koju će sadržavati u konačnoj, sortiranoj listi, tj. najmanju vrijednost u listi. Isti se postupak ponavlja za preostale elemente osim prvog. Postupak se ponavlja elementima od drugog prema posljednjem. U svakom sljedećem prolazu se dio liste koji treba obraditi smanjuje za jedan element. Zbog toga, ako je lista dugačka *n* elemenata, onda će, nakon *n-1* prolaza ovaj algoritam sortirati listu. [2]

Algoritam sortiranja izborom najmanjeg elementa će se u jeziku C realizirati sljedećom funkcijom selectionSort(). Ova funkcija kao argumente prima cjelobrojno polje ***a*** [ ] i njegovu duljinu ***n***. Ova funkcija mijenja polje ***a*** [ ] time što ono postaje sortirano. Pri sortiranju također postoji pomoćna funkcija swap () koja služi za zamjenu vrijednosti dviju cjelobrojnih varijabli sa zadanim adresama. Programski kod navedenog algoritma izgleda ovako:

```
void selectionSort(int a[], int n)
{
	int i, j, min;
	for (i = 0; i < n; i++) {
		min = i;
		for (j = i+1; j < n; j++) {
			if (a[j] < a[min]) {
				min = j;
				swap(&a[i], &a[min]);
			}
		}
	}
}
```

[2] Divjak, Blaženka; Lovrenčić, Alen. Diskretna matematika s teorijom grafova. Varaždin: TIVA-FOI, 2005.

```
void swap(int *x, int *y)
{
	int aux;
	aux = *x;
	*x = *y;
	*y = aux;
}
```

Pri analiziranju vremenske složenosti i algoritma sortiranja izborom najmanjeg elementa (engl. *selection sort*) bavimo se sljedećim: kada dođemo do prvog pronalaska, imamo n – 1 usporedbi. Pri svakom sljedećem pronalasku taj broj usporedbi smanjivat će se za 1. Na osnovu toga može se zaključiti da ukupni broj usporedbi iznosi

$$(n - 1) + (n-2) + (n-3).... +2 + 1= n (n-1) /2.$$

Obzirom da se u svakom pronalasku nalazi još jedna zamjena, imamo još 3 (n-1) operacija pridruživanja. Na osnovu toga zaključujemo da je ukupni broj operacija

$$n (n -1) /2 +3 (n-1) = O (n^2)$$

Ideja *selection sort*-a jest koristiti usporedbe i zamjene elemenata u nizu. Dakle cilj je dovesti najmanji element niza na njegovo mjesto. To mjesto je prvo u cijelom nizu, stoga je nova vrijednost elementa nakon zamjene upravo najmanji element niza. Postupak se ponavlja na nesređenom, skraćenom nizu za n-1. Možemo zaključiti da se niz skraćuje sprijeda. Ovaj postupak se ponavlja dok se ne dođe do niza sa samo jednim elementom, jer taj niz zasigurno možemo smatrati sortiranim.

4. Mjehuričasto sortiranje

Pri mjehuričastom sortiranju (engl. *bubble sort*) prolazi se redom po elementima liste i svaki se od njih uspoređuje sa svojim sljedbenikom. Ako je neki element veći od svoga sljedbenika, onda im se zamjenjuju mjesta. Nakon prvog prolaza na kraju liste pokazat će se najveća vrijednost. Nakon toga se na listi postupak ponavlja za prvih *n-1* elemenata pa će nakon prolaza kroz listu u pretposljednji element doći druga najveća vrijednost. Tako će nakon *n-1* prolaza lista biti sortirana. [3]

Ideja *bubble sort*-a jest prolaz kroz niz unaprijed od početka do kraja. Ako dva susjedna elementa nisu u dobrom poretku zamijenuje se njihova vrijednost odnosno mjesto. Pri dolasku do kraja niza postupak se ponavlja. Obzirom da se stalno vraćamo na početak, nije jasno kada se točno s postupkom staje.

Funkcija bubbleSort implementira algoritam sortiranja tako što će zamijeniti susjedne elemente u jeziku C. Funkcija prima kao argumente cjelobrojno polje ***a*** [] i njegovu duljinu ***n***, te mijenja polje ***a*** []. Ista pomoćna funkcija swap () se koristi za zamjenu vrijednosti dviju cjelobrojnih varijabli kao i kod *selection sort-a*. Programski kod navedenog algoritma izgleda ovako:

```
void bubbleSort(int a[], int n)
{
	int i, j;
	for (i = 0; i < n-1; i++) {
		for (j = 0; j < n-1-i; j++) {
			if (a[j+1] < a[j]) {
				swap(&a[j], &a[j+1]);
			}
		}
	}
}
```

[3] Divjak, Blaženka; Lovrenčić, Alen. Diskretna matematika s teorijom grafova. Varaždin: TIVA-FOI, 2005.

Implementacija pri sortiranju zamjenom susjednih elemenata se može poboljšati na način da se postupak zaustavi ako se ustanovi da u nekom od prolaza nizom nije došlo do zamjene susjednih elemenata. U nastavku će biti prikazana naprednija verzija bubbleSort () funkcije.

```
void bubbleSortAdv(int a[], int n)
{
	int i, j, chg;
	for (i = 0, chg = 1; chg; i++) {
		chg = 0;
		for (j = 0; j < n-1-i; j++) {
			if (a[j+1] < a[j]) {
				swap(&a[j], &a[j+1]);
				chg = 1;
			}
		}
	}
}
```

Pri analizi vremenske složenosti navedenog algoritma sortiranja na način zamjene susjednih elemenata dolazimo do zaključaka da u prvom prolazu imamo n-1 usporedbi i n-1 zamjena elemenata. U drugom prolazu u najgorem slučaju imamo n-2 usporedbi i n-2 elemenata. Kada dođemo do (n-1) prolaza u najgorem scenariju imat ćemo jednu usporedbu i jednu zamjenu elemenata. Izračunima dolazimo do zaključka da u najgorem slučaju ukupan broj operacija iznosi:

$$4 (n-1) + 4 (n-2) .... + 4*1 = 4n (n-1) / 2 = 2n(n-1).$$

Može se zaključiti kako je kompleksnost ovog algoritma i u prosječnom i u najgorem slučaju $O(n^2)$, gdje je n broj elemenata koji se sortiraju.

5. Sortiranje umetanjem

Ovaj način sortiranja jedan je od osnovnih algoritama sortiranja. Temeljen je na idejama dijeljenja liste na dva dijela: prvi koji je sortiran i drugi koji će se sortirati. U prvom dijelu nalazi se samo jedan element liste. U svakom sljedećem koraku se uzima prvi element iz drugog dijela liste i umeće se na odgovarajuće mjesto u prvom dijelu. Ako je drugi element manji od prvoga, prvi i drugi element mijenjaju mjesta. Nakon toga uzima se treći element i uspoređuje s drugim. Isti se postupak primjenjuje za četvrti element i tako redom. Pri nailasku na element koji je manji od elementa koji je promatran, promatrani element pomiče se za jedno mjesto udesno. Kada se svi elementi obrade na ovaj način lista elemenata bit će sortirana.

Implementacija jednostavnog sortiranja umetanjem (engl. *insertion sort*) u jeziku C svodi se na sljedeću funkciju insertionSort(). Funkcija opet kao argument prima cjelobrojno polje ***a*** [ ] kojeg treba sortirati, te njegovu duljinu *n*, a programski kod izgleda ovako:

```
void insertionSort(int a[], int n)
{
	int i, j, aux;
	for (i = 1; i < n; i++) {
		aux = a[i];
		for (j = i-1; j >= 0 && a[j] > aux; j--) {
			a[j+1] = a[j];
		}
		a[j+1] = aux;
	}
}
```

Kada analiziramo vremensku složenost algoritma za sortiranje umetanjem, dolazimo do zaključka da u k-tom prolasku unatrag prolazimo sortiranim dijelom polja duljine ***k***. Elemente koje tamo vidimo pomjeramo za jedno mjesto dalje sve dok su oni veći od elementa kojega želimo umetnuti na pravo mjesto. [4]

[4] Manger, Robert, Marušić, Miljenko. Strukture podataka i algoritmi, 2007. URL: http://web.studenti.math.pmf.unizg.hr/~manger/spa/skripta.pdf (2012-08-01)

Na osnovu navedenih informacija zaključujemo da bi ukupni broj operacija u najgorem slučaju bio:

$$2 * 1 + 2 * 2 + \ldots + 2 (\boldsymbol{n} - 1) = \boldsymbol{n} (\boldsymbol{n} - 1)$$

Veličina za vrijeme izvođenja je opet $O(n^2)$. Usprkos takvoj asimptotskoj ocjeni, ovaj algoritam se u praksi ipak pokazuje bržim od prije opisanih algoritama sa zamjenom elemenata.

6. Rekurzivni algoritmi za sortiranje

Rekurzivnim algoritmima smatramo one algoritme koji pozivaju same sebe sve dok ne postignu određen uvjet. Rekurzivni algoritmi su vrlo često usko vezani uz implementaciju pojedine matematičke funkcije na primjer Fibbonacijeve funkcije. Svaka rekurzija mora imati uvjet zaustavljanja koji će joj omogućiti izlaz. Svaki poziv rekurzije mora se približavati uvjetu zaustavljanja. Rekurzivna rješenja su kraća, ali je za njihovo izvođenje potrebno više vremena i memorije.

Svaka rekurzija mora imati uvjet zaustavljanja koji će omogućiti izlazak iz rekurzije. U protivnom rekurzivna funkcija će se izvoditi beskonačno. Također, svaki poziv rekurzije mora se približavati uvjetu zaustavljanja.

U ovom radu bit će obrađena dva rekurzivna algoritma za sortiranje, sortiranje pomoću sažimanja, (engl. *merge sort*) i brzo sortiranje, (engl. *quick sort*).

Sličnost ovih dvaju algoritama sortiranja je ta što i jedan i drugi dijele polje koje je zadano u dva manja polja, a zatim se rekurzivnim pozivima sortiraju ta dva mala polja te na kraju spajaju manja sortirana polja u jedno sortirano polje.

Razlika između ova dva algoritma sortiranja je u načinu na koji se dijeli veliko polje i načinu na koji se spajaju manja sortirana polja u veće. Upravo zbog ovih razlika, ova dva rekurzivna algoritma sortiranja imaju drugačije osobine što se tiče vremenske složenosti.

6.1. Sortiranje spajanjem

Sortiranje spajanjem (engl. *merge sort*) je prvi algoritam u ovom radu opisan sa složenosti manjom od kvadratne. Složenost ovog algoritma jednaka je $O(n \lg n)$. Algoritmi s ovom vrstom složenosti najčešće koriste metodu *podijeli pa vladaj* (lat. *divide et imepera*). Ova vrsta sortiranja koristi algoritam za spajanje dvije sortirane liste u jednu također tako sortiranu listu. U slučaju da se zadano polje sastoji samo od jednog elementa, ono je već sortirano.[5] Inače se zadano polje dijeli na dva manja podjednaka polja. Ta dva polja zatim se zasebno sortiraju rekurzivnim pozivima istog algoritma. Ta mala polja sažimaju se u jedno sortirano polje uz pomoć prethodno opisanog postupka sažimanja.

Ovaj algoritam se može lako implementirati u jeziku C kao kombinacija funkcija mergeSort() i merge(). Funkcija se poziva sa sljedećim argumentima: mergeSort(array, 0, n - 1), gdje je n broj elemenata polja koje je potrebno sortirati, a programski kod izgleda ovako:

```
void mergeSort(int *array, int left, int right)
{
    int mid = (left + right) / 2;
    if (left < right) {
            mergeSort(array, left, mid);
            mergeSort(array, mid + 1, right);
            merge(array, left, mid, right);
      }
}


void merge(int *array, int left, int mid, int right)
{
      int aux[right - left + 1];
      int pos = 0, l_pos = left, r_pos = mid + 1;
      while (l_pos <= mid && r_pos <= right) {
            if (array[l_pos] < array[r_pos]) {
                  aux[pos++] = array[l_pos++];
            } else {
                  aux[pos++] = array[r_pos++];
            }
      }
      while (l_pos <= mid) {
            aux[pos++] = array[l_pos++];
      }
      while (r_pos <= right) {
            aux[pos++] = array[r_pos++];
      }
```

[5] Divjak, Blaženka; Lovrenčić, Alen. Diskretna matematika s teorijom grafova. Varaždin: TIVA-FOI, 2005.

```
	int iter;
	for (iter = 0; iter < pos; iter++) {
		array[iter + left] = aux[iter];
	}
	return;
}
```

Prednost ovog načina sortiranja u odnosu na druge algoritme jest mogućnost sortiranja velikih polja pohranjenih u vanjskoj memoriji računala. Što se tiče nedostataka u odnosu na druge tzv. brze algoritme je dodatno trošenje memorije koje je potrebno zbog prepisivanja polja tijekom sažimanja.

6.2. Brzo sortiranje

Pokušaj ubrzanja faze spajanja doveo je do potpuno novog algoritma sortiranja, tzv. *quicksort* algoritma. Ovaj algoritam također se temelji na metodi *podijeli pa vladaj*, no kod njega je faza spajanja potpuno izbjegnuta. Prvo je potrebno istaknuti jedan element liste- tzv. pivot ili stožer element. Za pivot element može se uzeti prvi element liste, srednji između prva tri ili ga se može izračunati na bilo koji drugi način. Važno je da se pivot mora izračunati u vremenu O (1). Nakon toga se elementi preslaguju tako da u prvom dijelu liste budu elementi koji su manji od pivota, a u drugom dijelu liste elementi veći od pivota. Poslije se algoritam izvodi rekurzivno i za prvi i drugi dio liste. Postupak se ponavlja sve dok lista nema manje od dva elementa, a tada sortiranje postaje trivijalno. [6]

Algoritam brzog sortiranja (engl. *quick sort*) se može u jeziku C implementirati na više načina jer postoje razne mogućnosti izbora stožera. Najjednostavnija varijanta implementacije je funkcija quickSort ( ). Pri ovoj funkciji se u svakom rekurzivnom pozivu kao početni stožer bira početni element pod-polja koji je objekt sortiranja. Da bi polje ***a*** [ ] duljine ***n*** moglo biti sortirano, program koji je glavni mora pozvati quickSort sa sljedećim argumentima: quickSort(a, 0, n - 1), a programski kod izgleda ovako:

```
void quickSort(int a[], int lower, int upper)
{
	int i;
	if (upper > lower) {
		i = split(a, lower, upper);
		quickSort(a, lower, i - 1);
		quickSort(a, i + 1, upper);
	}
}

int split(int a[], int lower, int upper)
{
	int i, p, q, t;
	p = lower + 1;
	q = upper;
	i = a[lower];
	while (q >= p) {
		while (a[p] < i) {
			p++;
		}
		while (a[q] > i) {
			q--;
		}
		if (q > p) {
```

[6] Divjak, Blaženka; Lovrenčić, Alen. Diskretna matematika s teorijom grafova. Varaždin: TIVA-FOI, 2005

```
                    t = a[p];
                    a[p] = a[q];
                    a[q] = t;
                }
            }
            t = a[lower];
            a[lower] = a[q];
            a[q] = t;
            return q;
    }
```

Analizom vremenske složenosti *quicksort* algoritma dolazimo do nekoliko zaključaka. Naime provjera da algoritam u najgorem slučaju ima složenost O (n^2) je vrlo jednostavna. Taj najgori slučaj se događa kada je stožer početni element te je polje već sortirano. Matematički je dokazivo da je prosječno vrijeme izvršavanja O (n log n).[7] Kako će se algoritam ponašati ovisi o tome na koji se način birao stožer. Mogućnosti za biranje stožera su često početni element, odnosno medijan koji je izabran između tri elementa.

[7] Quick sort. URL: http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Tremblay/L06-QuickSort.htm (2012-07-03)

7. Sortiranje pomoću binarnih stabala

Binarno stablo je stablo kod kojeg svaki čvor može imati najviše dva nasljednika. Oni su lijevi i desni nasljednik. Opće stablo može se prikazati binarnim stablom tako da za svaki čvor prvi nasljednik postane lijevi, a prvi susjed postane desni nasljednik u binarnom stablu. Stablo u kojem su sve razine popunjene a jedino posljednja razina ne mora biti popunjena zove se potpuno binarno stablo. Svaka razina popunjava se s lijeva na desno. Ako je još zadovoljen uvjet, da je nadređeni čvor veći ili jednak od oba podređena, takvo potpuno binarno stablo zove se hrpa (engl. *heap*). [8]

Od ove vrste algoritama za sortiranje obradit će se još dva sortirajuća algoritma, sortiranje obilaskom binarnog stabla traženja, (engl. *tree sort*) , odnosno sortiranje pomoću hrpe, (engl. *heap sort*) . Ova dva algoritma vrlo su slična, svaki od njih radi na način sortiranja polja tako da elemente polja najprije ubaci u binarno stablo, a zatim ih izvadi iz toga stabla u potpuno sortiranom redoslijedu. Ono u čemu se razlikuju je vrsta binarnog stabla koje se koristi. U slučaju *tree sort*-a je to binarno stablo traženja, a u slučaju *heap sort*-a je to hrpa.

[8] Papić, Anita. Binarno stablo. Diskretne strukture i algoritmi. Sveučilište J. J. Strossmayer, Filozofski fakultet, Odsjek za informacijske znanosti. Osijek, 8. 12.2011. [Predavanje].

7.1. Sortiranje pomoću hrpe

Sortiranje pomoću hrpe (engl. *heap sort*) za razliku od svih dosad navedenih vrsta algoritama, temelji se na posebnom tipu podataka – hrpi. Hrpom nazivamo binarno stablo sa svojstvom da vrh ima manju vrijednost od svoja dva nastavka. Hrpa se puni takvim redom da su joj gotovo sve razine (osim posljednje) popunjene potpuno. Posljednja se razina puni s lijeva na desno. Pri brisanju ili dodavanju elemenata treba obratiti posebnu pozornost na to da sva svojstva hrpe budu zadovoljena. Novi element uvijek se nadodaje na posljednjoj razini stabla, a nakon toga njegova se vrijednost uspoređuje s vrijednošću njegovog roditelja. Ako je vrijednost novog elementa manja od vrijednosti svog roditelja trebamo im zamijeniti mjesto. Ovaj postupak se ponavlja sve dok novonadodani element ne dobije roditelja koji je manji od njega, ili barem dok ne postane korijen stabla.[9]

Implementacija ovog algoritma obavlja se na način sortiranja polja a[ ] duljine n tako da se poziva funkcija heapSort ( ). Za uspješnu provedbu *quicksort* algoritma uz navedenu funkciju koristi se i pomoćna shiftDown() funkcija, a programski kod izgleda ovako:

```
void heapSort(int a[], int n)
{
	int i, aux;
	for (i = n/2; i >= 0; i--) {
		shiftDown(a, i, n - 1);
	}
	for (i = n-1; i >= 1; i--) {
		aux = a[0];
		a[0] = a[i];
		a[i] = aux;
		shiftDown(a, 0, i-1);
	}
}
```

[9] Divjak, Blaženka; Lovrenčić, Alen. Diskretna matematika s teorijom grafova. Varaždin: TIVA-FOI, 2005.

```
void shiftDown(int a[], int root, int bottom)
{
	int max_child = root*2 + 1;
	if (max_child < bottom) {
		int other_child = max_child + 1;
		max_child = (a[other_child] > a[max_child]) ? other_child : max_child;
	} else {
		if (max_child > bottom) {
			return;
		}
	}
	if (a[root] >= a[max_child]) {
		return;
	}
	int aux = a[root];
	a[root] = a[max_child];
	a[max_child] = aux;
	shiftDown(a, max_child, bottom);
}
```

Polje a [ ] koje je objekt sortiranja se interpretira kao prikaz potpunog binarnog stabla koje na početku nije hrpa. Pozivom pomoćne funkcije buildHeap ( ) oznake se premještaju tako što se zadovoljava svojstvo hrpe.

Kod analize vremenske složenosti *heap sort*-a zaključujemo da se algoritam svodi na n-struku primjenu operacije ubacivanja elementa u hrpu, tj. izbacivanja najmanjeg elementa iz hrpe. Stoga je vrijeme u najgorem slučaju za cijelo sortiranje O (n log n).[10] Eksperimentima je dokazivo da sortiranje pomoću hrpe zbilja spada među najbrže algoritme za sortiranje jer ima mogućnost velikom brzinom uspješno sortirati velika polja.

Bitno je spomenuti da je danas najbrži korišten algoritam za sortiranje *introspective-sort*. Taj hibridni algoritam je kombinacija dvaju već navedenih algoritama *quicksort* te *heapsort*. Naime, kada program detektira da se nekoliko puta za redom nailazi na loš raspored elemenata za *quick-sort* i da bi u tom slučaju došlo do kvadratnog očekivanog vremena izvođenja, on se prebacuje na *heap-sort* koji je u tom slučaju znatno bolje rješenje.[11]

[10] Heap sort. URL: http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/heap/heapen.htm (2012-07-03)

[11] Musser, Robert. Introspective Sorting and Selection Algorithms.// Software: Practice and Experience 27,8/(1997), str. 983-993.

7.2. Sortiranje obilaskom binarnog stabla traženja

Kada govorimo o algoritmu sortiranja obilaskom binarnog stabla traženja, (engl. *tree sort*) podrazumijevamo implementaciju na način kombiniranja funkcija za rad s binarnim stablima traženja, te funkcija za obilazak binarnih stabala. Međutim u dalje navedenom primjeru će biti navedena modificirana verzija koda u programiranju. Dakle, polja a [] duljine n se sortiraju pozivom naredbe treeSort (a, n). Funkcija treeSort( ) radi na način pozivanja triju pomoćnih funkcija, a to su: insert(), writeInorder() i destroy(). Kada uzastopno pozivamo insert () funkciju memorija se dinamički alocira te se gradi binarno stablo traženja u kome se nalaze svi podaci prepisani iz polja a []. Kada koristimo funkciju writeInorder () obavlja se obilazak već izgrađenog binarnog stabla, te se paralelno prepisuju podaci iz binarnog stabla natrag u polje u skladu s redoslijedom obilaska. Kod funkcije destroy () se razgrađuje binarno stablo koje alocira memoriju, a programski kod izgleda ovako: [12]

```
typedef struct cell tag
{
	int element;
	struct cell tag *left_child;
	struct cell tag *right_child;
} celltype;

void treeSort(int a[], int n)
{
	int i, next;
	celltype *tree;
	tree = NULL;
	for (i = 0; i < n; i++) {
		tree = insert(a[i], tree);
	}
	next = 0;
	writeInOrder(tree, a, &next);
	destroy(tree);
}
```

[12]Manger, Robert; Marušić, Miljenko. Strukture podataka i algoritmi, 2007. URL: http://web.studenti.math.pmf.unizg.hr/~manger/spa/skripta.pdf (2012-08-01)

```
celltype *insert(int x, celltype *node)
{
	if (node == NULL) {
		node = (celltype*) malloc(sizeof(celltype));
		node->element = x;
		node->left_child = node->right_child = NULL;
	} else if (x < node->element) {
		node->left_child = insert(x, node->left_child);
	} else {
		node->right_child = insert(x, node->right_child);
	}
	return node;
}

void writeInOrder(celltype *node, int a[], int *np)
{
	if (node != NULL) {
		writeInOrder(node->left_child, a, np);
		a[*np] = node->element;
		*np += 1;
		writeInOrder(node->right_child, a, np);
	}
}

void destroy(celltype *node)
{
	if (node != NULL) {
		destroy(node->left_child);
		destroy(node->right_child);
		free(node);
	}
}
```

8. Zaključak

Svi algoritmi koji su navedeni u ovom radu temelje se na usporedbi kao osnovnoj operaciji koja uvjetuje premještanje elemenata u polju. Kao što je već navedeno sortiranje je prisutno zbog olakšavanja raznih procesa kako u mnogim djelatnostima tako i u informatici. Algoritmi su niz naredbi koji efikasno rješavaju neki problem. Uz razvoj tehnologije razvijali su se i razni algoritmi sortiranja s ciljem pojednostavljenja samog postupka sortiranja. Danas se sortiranje može vidjeti u nekim od najosnovnijih operacija u računalu, npr. prikazu datoteka u direktoriju i sortiranja pjesama u glazbenim playerima. Kako je *bubblesort* vrlo spor i vrlo nepraktičan algoritam, stručnjaci su konstantno radili na poboljšanjima te su nastali razni novi jednostavni algoritmi kao što su *selection sort*, *insertion sort*, zatim algoritmi za sortiranje koji rade na osnovu rekurzije. Najbolji primjer je *quicksort*, jedan od najbržih algoritama opće namjene za sortiranje, i drugi. U zadnje vrijeme često se postavlja pitanje može li bolje. S obzirom na brz rast tehnologije zadnjih desetljeća možemo očekivati također i poboljšane algoritme za sortiranje na kojima se već radi. Jedan od primjera je u radu navedeni *introspective-sort* koji je hibrid između dvaju algoritama *quicksort* te *heapsort,* a obzirom da se sastoji od kombinacije najboljih osobina ovih dvaju algoritama, on predstavlja po mnogim stručnjacima današnji najbrži algoritam za sortiranje te najbolje rješenje za ovu problematiku. Naravno, s obzirom da se radi o računalnoj tehnologiji gdje ovi algoritmi imaju najveću primjenu, na razvoju boljih rješenja svakodnevno se radi te je moguće kako je 60-ih godina prošlog stoljeća *quicksort* algoritam zamijenio *bubble sort*, da će se pojaviti novi način za sortiranje podataka koji će se pokazati učinkovitijim od svih navedenih algoritama za sortiranje.

9. Literatura

1. Baumgartner, Alfonzo; Poljak, Stjepan. Sortiranje podataka. Osijek, 2005.

2. Bubble sort. URL: http://www.sorting-algorithms.com/bubble-sort (2012-08-01)

3. David R. Musser. Introspective Sorting and Selection Algorithms. Troy,1997.

4. Divjak, Blaženka; Lovrenčić, Alen. Diskretna matematika s teorijom grafova. Varaždin, 2005.

5. Heap sort. URL: http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/heap/heapen.htm (2012-07-03)

6. Heap sort. URL: http://rosettacode.org/wiki/Sorting_algorithms/Heapsort (2012-07-03)

7. Insertion sort. URL: http://programminggeeks.com/c-code-for-insertion-sort/ (2012-07-03)

8. Insertion sort. URL: http://www.programmingsimplified.com/c/source-code/c-program-insertion-sort (2012-07-03)

9. Manger, Robert; Marušić, Miljenko. Strukture podataka i algoritmi. Zagreb: Prirodoslovno matematički fakultet, 2007.

10. Merge sort. URL: http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/merge/mergen.htm (2012-07-03)

11. Papić, Anita. Binarno stablo. Diskretne strukture i algoritmi. Sveučilište J. J. Strossmayer, Filozofski fakultet, Odsjek za informacijske znanosti. Osijek, 8. 12.2011. [Predavanje].

12. Selection sort. URL: http://www.algolist.net/Algorithms/Sorting/Selection_sort (2012-07-03)

13. Selection sort. URL: http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm (2012-07-03)

14. Tree sort. URL: http://chinmaylokesh.wordpress.com/2012/02/16/tree-sort-in-order-traversal-of-a-binary-search-tree/ (2012-07-03)

15. Quick sort. URL: http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Tremblay/L06-QuickSort.htm (2012-07-03)